PowerEnJoy

Politecnico di Milano

Industrial and Information Engineering Computer Science and Engineering

Cattaneo Davide

El Hariry Matteo

Frontino Francesco

Integration Test Plan Document

Contents

1 Introduction

1.1 Purpose and scope

Purpose of this document is to define an approach for testing software during the development phase. The integration and test plan document (ITPD) provides guidelines for the developers about which components of the software platform are to be tested and how to test them.

Both integration and standalone module testing methodologies are reported in this document.

1.2 Definitions, acronyms and abbreviations

Here is a brief description of the most important actors and words used in our system:

- **System:** is the application core. The software system which will perform all the operations and monitor interactions and be a medium between users and cars.
- **Operator:** is a flexible actor in our system. He's part of a set of people operating under the administrator directions. Their normal tasks are to bring to charging stations cars left with less than 15% battery level, interact with users which call for help during a ride, intervene when necessary (e.g. a wheel brakes during a ride). Their exceptional task can be the case in which they have to go and get back cars taken by the police or cars involved in incidents etc.
- **Administrator:** the administrator of the system is the person allowed to manage eventual unexpected cases (like incidents and damaging situations). He is the person notified every time a problem occurs, and once analyzed the situation (s)he'll decide how to handle it (call for support, send operators, call the police etc.).

- **Board Controller:** BC is the car system, which includes all the hardware and software components interacting with the vehicle itself, with the users' smartphone and the Central System. The Main components part of the BC are the CAN controller, the Android System and the car display. All of them are interconnected in order to guarantee an efficient flow of information from and to sensors and with the outside environment (users and central system).
- **CAN:** A Controller Area Network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts.

1.3 Reference documents

Specification Document:
* Assignments 1 and 2 (RASD and DD).pdf
* Verification and validation, part I and II
* Verification Tools

Examples documents:
* Sample Integration Test Plan Document
* Integration testing example document

2 Integration strategy

2.1 Overview

To provide a correct, robust and resilient software, the test plan aims to ensure that at least 80% of the application code results to be covered by unity testing. At this purpose, this sections provides the major guidelines to follow during the integration and testing phase. Those reported in this document represent the minimum set of test for guaranteeing a sufficient level of correctness and robustness.

2.2 Elements to be integrated

Starting from the previous releases of the project (RASD and DD) these high-level core modules can be identified in the system:

- the Database System
- the Central System (application core)
- the Board Controller (car system)
- the User App
- the Administration System
- the Notifier
- the Payment Gateway

In particular, these are the high-level components associated with each module that must be tested:

1) the Database System
    - DBMS
2) the Central System (application core)
    - UserManager
    - CarManager
    - RideManager

3) the Board Controller (car system)
    - RemoteController
    - CANcontroller
    - RideFunctions
    - MapsAPI
    - AssistanceRequest
4) the User App
    - AccountFunctions
    - MapsAPI
5) the Administration System
    - MaintenanceFeautures
    - MapsAPI
6) the Notifier
7) the Payment Gateway

As a first step a low level of integration among the components of each module can be described.

1. Starting from the Database, the goal is to develop a solid and reliable platform for saving and retrieving data, such as the electric-cars being part of the sharing system, the users accounts and so on.

2. Now the focus must be on the Central System, which is the core of all the interactions of the PowerEnJoy reality. Here some more details on how each component is constructed to be provided:
   - UserManager
   - CarManager
   - RideManager

3. Board Controller is the following subsystem to be tested, this in order to evaluate the efficiency of the car system components and their connectivity with the environment and the headquarter (CS).
   First a test of the integration with the CAN controller has to be performed so that the connection with the car sensors, actuators and the whole electronic system of the vehicle is checked to work properly and the low operations are effective when executed.

The second inner integration is with the Map Service where the GPS system is here tested to respond and allowing the whole interaction with the navigation and map generations.

4. Once the previous modules are exanimated the User App can be assessed by checking its working behavior. In our specific case, this involves the integration of the AccountFunctions and MapsAPI service. The latter test is meant to make sure that the user application is able to precisely use the map service, which is essential to let users find the available cars near their location or in the address provided.

5. For what concerns the building of the Administration System the components involved are the Maintenance Features and MapsAPI. The first is meant to allow both administrators and operators to perform tasks through the browser portal accessible with their credentials, the second is useful to let both admins and operators to find cars location.

6. (7.) Finally, two commercial, already existing components are used to achieve specific functionalities: Notification System and Payment Gateway. They must be tested by checking if, for the Notifier case, all kind of messages the system uses for information exchange are correctly forwarded through the different agents, and, for the Payment Gateway, if it can correctly receive requests and respond by giving confirmation of the operation outcome.

All the above components and subsystems specifically developed for PowerEnJoy can be seen using the architectural client/server point of view:

– On the server side: Ride Management System, System Administration and Account Management subsystems component.

– On the client side: Administration Web Application, User Web Application, User Mobile Application and the Board Controller (Car system manager) components.

2.3 Integration testing strategy

To test these modules and their interaction a bottom-up approach will be implemented.
Bottom up testing will be conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module.

Advantages:

- Advantageous if major flaws occur toward the bottom of the program.
- Test conditions are easier to create.
- Observation of test results are easier.

The strategy for testing and integration consists of 4 phases:

1. Testing the application core
2. Testing maintenance logic
3. Test the app interaction with the system
4. Test the car interaction with the system

These 4 phases are to be considered separately.

1) Testing the application core:

This first phase consists of:

1. Testing each single element reported in paragraph 2.2 (except maintenance manager) via unity testing
2. Integrating all the elements using a big bang strategy
3. Test the whole system core

2) Testing maintenance logic:

The second phase consists of:

1. Test the MaintenanceManager via unity testing
2. Integrate the MaintenanceManager in the system core
3. Test maintenance requests

3) Test the app interaction with the system:

The third phase consists of:

1. Test the application via unity testing
2. Test the interaction with the core system

4) Test the car interaction with the system:

The fourth phase consists of:

1. Test the application via unity testing
2. Test the interaction with the system core

This strategy aims at achieving a gradual integration by first testing the main logic, simulating every possible command coming from the "world", and then adding step by step a new element that replaces the precedent stubs to verify whether the real components act like they were meant to do.

Given that JEE is a tied framework, assumptions about the correctness of data storage and communications between layers are made. Moreover, HTTP requests for maintenance, reservation etc. are considered always working due to the robustness of the protocols.

The platform relies on two external services (payment gateway and Google APIs) that can't be tested and are assumed to be fully working for the system purpose.

2.4 Integration sequence

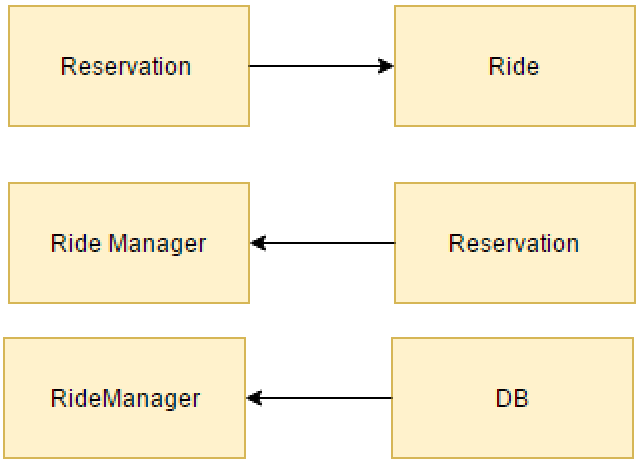
Following the already mentioned bottom-up approach, we now describe how the various subcomponents are integrated together to create higher level subsystems.

All the low level components in the system will be integrated according to the following diagrams and descriptions.

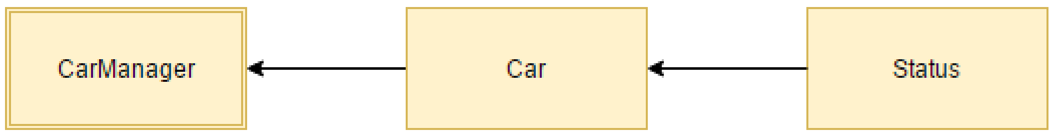
❖ <u>Database System</u>
The integration test for the database system consist of checking if the functionalities provided by the DB component (an already existing module to be simply used) are correctly working and providing an efficient storage and retrieval of data.

❖ <u>Central System subsystems integration:</u>

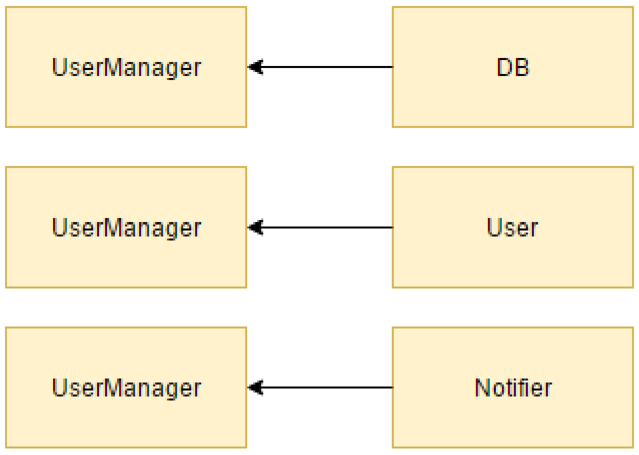
- RideManager component:
  - By integrating the reservation with the ride it can be checked if the two instances communicate properly (e.g. an allocation of a ride instance must start once a reservation is completed).
  - The same operation is done with ride manager and reservation.
  - Finally, the ride manager is integrated with the database, checking that all the operations needed to be saved will correctly update the database.



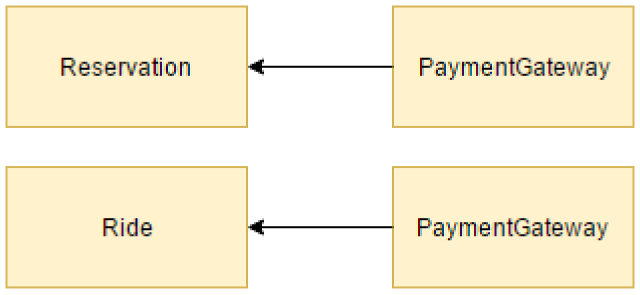
- CarManager component:
  The Car Manager instance is the core of this subsystem and it needs the integration with the car and with the status subcomponents.



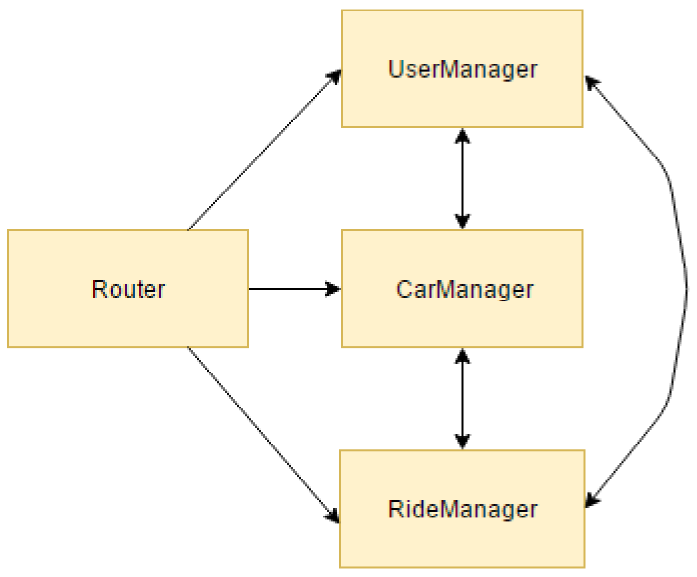
- UserManager component:



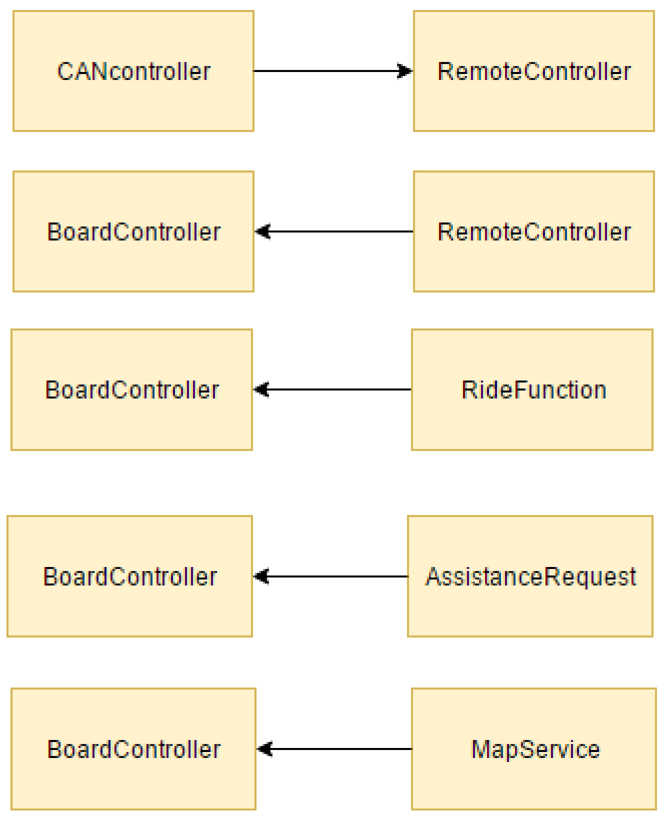
- An integration of the ride and reservation modules must be performed with the payment gateway



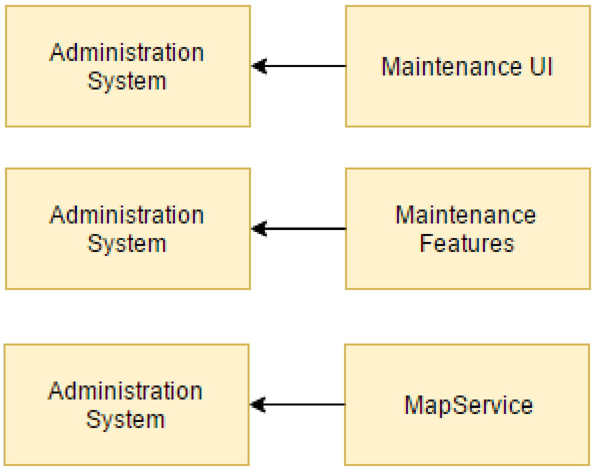
- The main module is now developed and a temporary program called DRIVER is used to simulate it as follows.
  Through the Router accesses the right connectivity and exchange of data between the three main subsystems with the outside environment can be checked. Also the correct workflow between the inner components themselves is tested here.



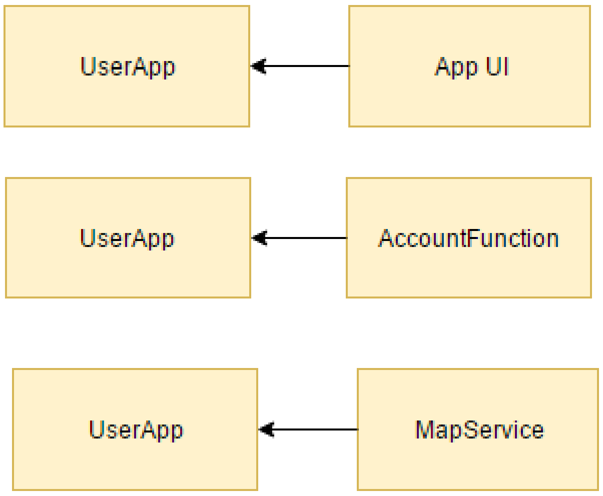
❖ Board Controller subsystems integration:



❖ Administration System components integration:



❖ User App components integration:



3 Tests description

**The notation X -> Y, used in the field "Environmental needs", is used to express that the X and Y must already be integrated*

3.1 User application tests description

Here are reported in detail all the tests aimed to cover the user application as for regards all the functions provided by the account functions interface.

3.1.1.1 User registration

Input	*User's data*
Output	*A new user has been created and store into the DB*
Environmental needs	*UserApp –> AccountFunctions*
Exceptions	• *Invalid user data* • *Username already taken*

3.1.1.2 Update information

Input	*New user's data*
Output	*User's data have been updated*
Environmental needs	• *The user's account must exist* • *UserApp –> AccountFunctions*
Exceptions	• *Invalid user data*

3.1.1.3 Delete account

Input	-
Output	*User's account has been deleted and marked as "dismissed" in the DB*
Environmental needs	• *The user's account must exist* • *UserApp –> AccountFunctions*
Exceptions	-

3.1.1.4 User login

Input	*Username and password*
Output	*User logged in*
Environmental needs	• *The user's account must exist* • *UserApp –> AccountFunctions*
Exceptions	*Wrong Datas*

Here are reported in detail all the tests aimed to cover the user application as for regards the most representative functions provided by the account functions interface for reservation control.

3.1.1.5 Find available car

Input	• *User's position* • *Address*
Output	*A list containing all the available cars in the nearby of the given location*
Environmental needs	• *The user's account must exist* • *UserApp –> AccountFunctions*
Exceptions	• *Location access error*

3.1.1.6 Reserve car

Input	*A car selected from the list of the available cars*
Output	*A reservation has been created and added into the DB*
Environmental needs	• *The user must have looked for a car* • *UserApp –> AccountFunctions* • *An available car must have been found*
Exceptions	-

3.1.1.7 Unlock car via Bluetooth

Input	*The key related to the car to be unlocked*
Output	*The car results to be unlocked and a ride is started*
Environmental needs	• *The car must be reserved* • *UserApp –> AccountFunctions* • *BoardController –> RideFunctions*
Exceptions	• *Access denied* • *Out of range*

3.1.1.8 Unlock car via NFC

Input	*The code related to the car to be unlocked*
Output	*The car results to be unlocked and a ride is started*
Environmental needs	• *The car must be reserved* • *UserApp –> AccountFunctions* • *BoardController –> RideFunctions*
Exceptions	• *Access denied* • *Out of range*

3.2 Administration system tests description

Here are reported in detail all the tests aimed to cover the administration system as for regards all the functions provided by the maintenance features interface.

3.2.1 Unlock car admin

Input	-
Output	*The car is unlocked and the user has the full control of every car functionality*
Environmental needs	*AdministrationSystem -> MaintenanceFeatures*
Exceptions	• *Access denied due to lack of permissions* • *Out of range*

3.2.2 Assign task

Input	-
Output	*A task has been assigned to an operator and a notification is sent both to the asker and the employee*
Environmental needs	• *AdministrationSystem -> MaintenanceFeatures* • *The notifier must work*
Exceptions	• *No employee found*

3.2.3 Notify user

Input	*The message from the administrator*
Output	*A message has been sent to the user*
Environmental needs	• *AdministrationSystem -> MaintenanceFeatures* • *The notifier must work*
Exceptions	-

3.2.4 Update cars status

Input	*The ID of the car which status has to be updated*
Output	*The car status of the selected car has been changed*
Environmental needs	• *AdministrationSystem -> MaintenanceFeatures*
Exceptions	• *Inconsistent change*

3.3 Board controller tests description

Here are reported in detail all the tests aimed to cover the on board controller as for regards all the functions invoked by the user through the car screen.

3.3.1 Ride functions

3.3.1.1 Send status

Input	-
Output	*The BC sends the status of the vehicle to the central system*
Environmental needs	• *BoardController -> RideFunctions*
Exceptions	-

3.3.1.2 Park

Input	*Position and type of parking selected by the user*
Output	*The BC sends the park request to the central system and waits for an answer to show the user the outcome*
Environmental needs	• *BoardController -> RideFunctions*
Exceptions	*Incompatible location*

3.3.1.3 Safe mode

Input	-
Output	*The BC sends the safe mode request to the central system and waits for an answer to show the user where and how to park the vehicle to obtain bonuses*
Environmental needs	• *BoardController -> RideFunctions*
Exceptions	• *Unsatisfied minimum requirements*

3.3.2 MapsAPI

3.3.2.1 Navigation

Input	-
Output	*The BC keeps the navigation on the screen.*
Environmental needs	• *BoardController -> MapsApi*
Exceptions	• *Gps lost*

3.3.3 Remote car

Here are reported in detail all the tests aimed to cover the on board controller as for regards all the functions invoked by the central system to the car. Just low level control operations are involved in these operations.

3.3.3.1 Unlock car

Input	*The ID of the car to be unlocked*
Output	*The car has been unlocked*
Environmental needs	• *The car must be locked* • *CentralSystem -> RemoteCar* • *BoardController -> CANController*
Exceptions	• *Car already unlocked*

3.3.4 Assistance request

Tests related to an assistance request

3.3.4.1 Exceptional assistance request

Input	*The user's message: text or voice*
Output	*An assistance request has been sent to the administrator*
Environmental needs	• *The user must have accessed the vehicle* • *BoardController -> AssistanceRequest*
Exceptions	-

3.3.4.2 Routine assistance request

Input	*The routine ID*
Output	*An assistance request has been sent to the system*
Environmental needs	*BoardController -> AssistanceRequest*
Exceptions	-

3.4 Central system tests description

Here are reported in detail all the tests aimed to cover the main methods not yet tested but performed by the central system.

3.4.1.1 Grant permission

Input	*User ID, Permits required*
Output	*The request is accepted or rejected*
Environmental needs	• *Database integration*
Exceptions	-

3.4.1.2 Calculate costs

Input	*The list of the travels made by the user*
Output	*A total cost for the ride*
Environmental needs	*The ride must be concluded*
Exceptions	-

3.4.1.3 Verify bonuses

Input	*The list of the travels made by the user and their statuses*
Output	*The total discount*
Environmental needs	*The ride must be concluded*
Exceptions	-

4 Tools and equipment

4.1 Tools

Testing procedures are to be executed using one of the following operating systems:

- Windows 10
- MacOS 10.12
- Linux 4.8

due to guarantee the application to be working in the most modern environments.
Both the car application and the system core must be developed using the latest Eclipse IDE version and must be based on Java 7 APIs. The framework used to develop the business logic must be JEE 7.
Junit is the only framework suggested for testing the applications. The use of other frameworks aiming at improving the quality of testing is accepted.
To develop the web site, is mandatory to use HTML 5 / CSS 3 / JS primitives using Bootstrap as a base layer. No peculiar IDE nor editor is suggested in this case.

4.2 Equipment

To achieve a full testing, the following equipment is needed:

- Android smartphone (O.S. v4.4 or greater)
- An iOS smartphone (O.S. v7.0 or greater)
- A Windows Mobile smartphone (O.S. v10.0 or greater)
- An ARM mobile development board provided with Bluetooth, mobile connection, GPS and NFC modules.

For each smartphone is necessary to test the mobile application and the web site.
The ARM development platform used for testing the car application must be compliant with the specifications previously given (RASD).
No peculiar hardware is suggested for each element of the equipment but the following minimum requirements must be granted:

- Mobile devices:
  - 1.8 GHz CPU or above
  - 4G connection
  - Bluetooth 4.2 or above
- Development board:
  - 1.8 GHz CPU or above
  - High precision GPS

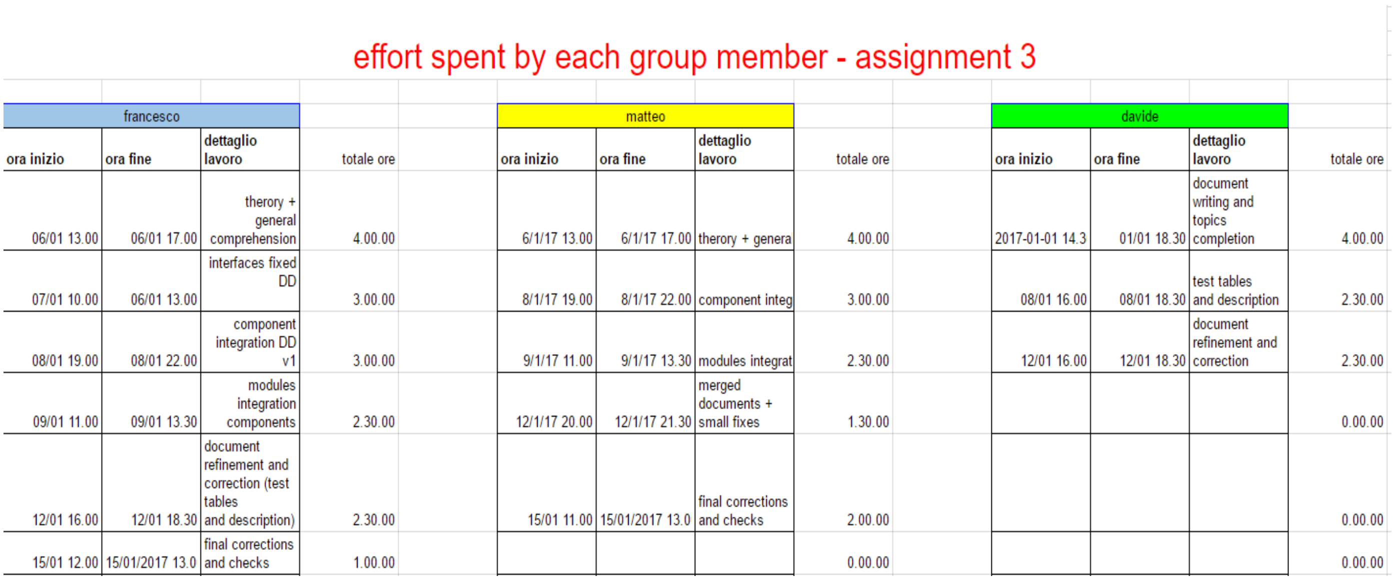
Due to the web site to run on desktop / laptop machines too, it is enough for it to be fully usable at least on two of the machines used for the development. No peculiar hardware is needed for the PCs.

5 Revision

5.1.1 Team work

Here is reported a compact table showing how the work was brought on by all the members of the group.

Document work finished on the 15th January 2017.



effort spent by each group member - assignment 3

francesco				matteo				davide			
ora inizio	ora fine	dettaglio lavoro	totale ore	ora inizio	ora fine	dettaglio lavoro	totale ore	ora inizio	ora fine	dettaglio lavoro	totale ore
06/01 13.00	06/01 17.00	therory + general comprehension	4.00.00	6/1/17 13.00	6/1/17 17.00	therory + genera	4.00.00	2017-01-01 14.3	01/01 18.30	document writing and topics completion	4.00.00
07/01 10.00	06/01 13.00	interfaces fixed DD	3.00.00	8/1/17 19.00	8/1/17 22.00	component integ	3.00.00	08/01 16.00	08/01 18.30	test tables and description	2.30.00
08/01 19.00	08/01 22.00	component integration DD v1	3.00.00	9/1/17 11.00	9/1/17 13.30	modules integrat	2.30.00	12/01 16.00	12/01 18.30	document refinement and correction	2.30.00
09/01 11.00	09/01 13.30	modules integration components	2.30.00	12/1/17 20.00	12/1/17 21.30	merged documents + small fixes	1.30.00				0.00.00
12/01 16.00	12/01 18.30	document refinement and correction (test tables and description)	2.30.00	15/01 11.00	15/01/2017 13.0	final corrections and checks	2.00.00				0.00.00
15/01 12.00	15/01/2017 13.0	final corrections and checks	1.00.00				0.00.00				0.00.00